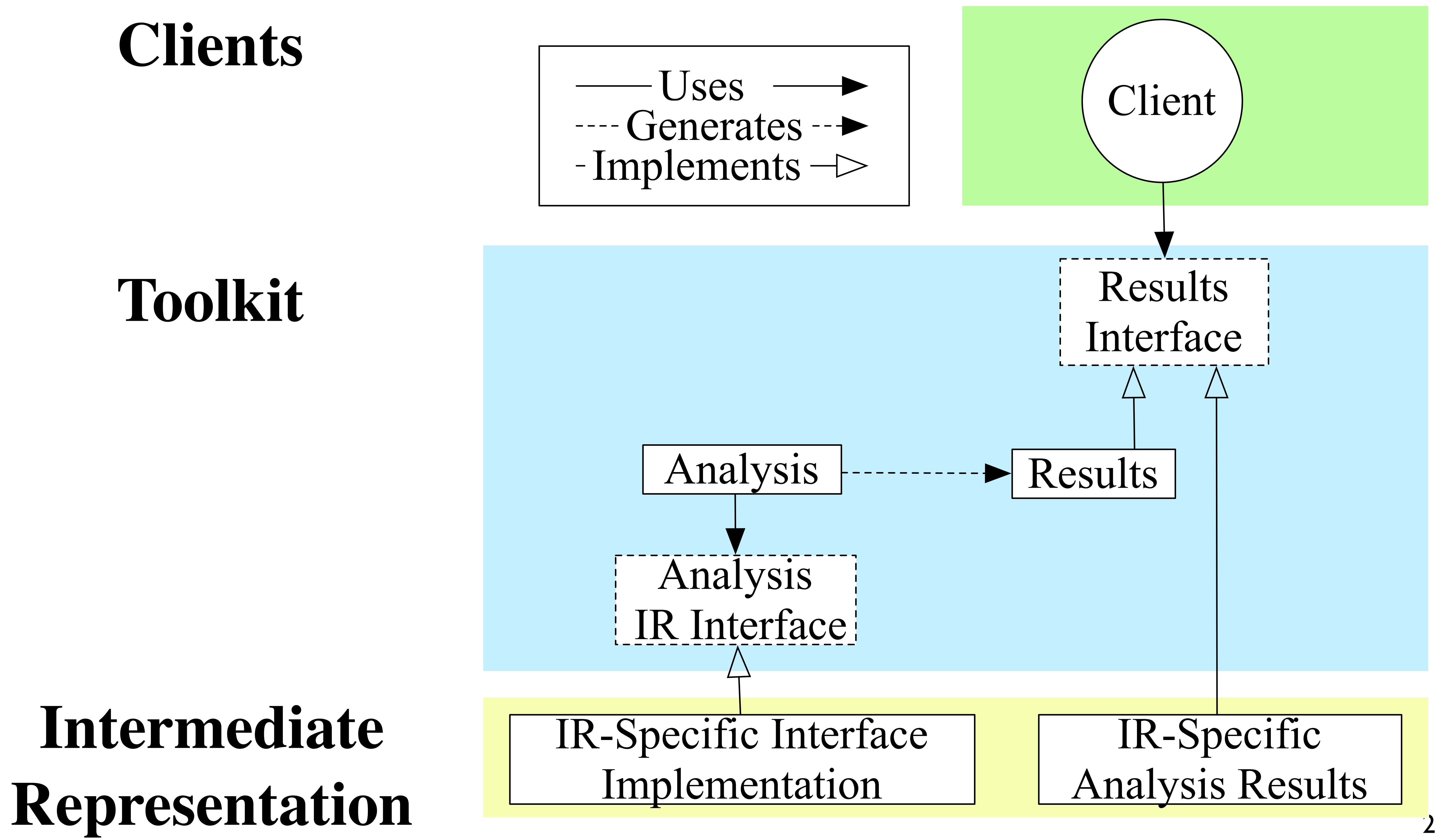
OpenAnalysis: Status as Used in OpenADFortTk and ADIC

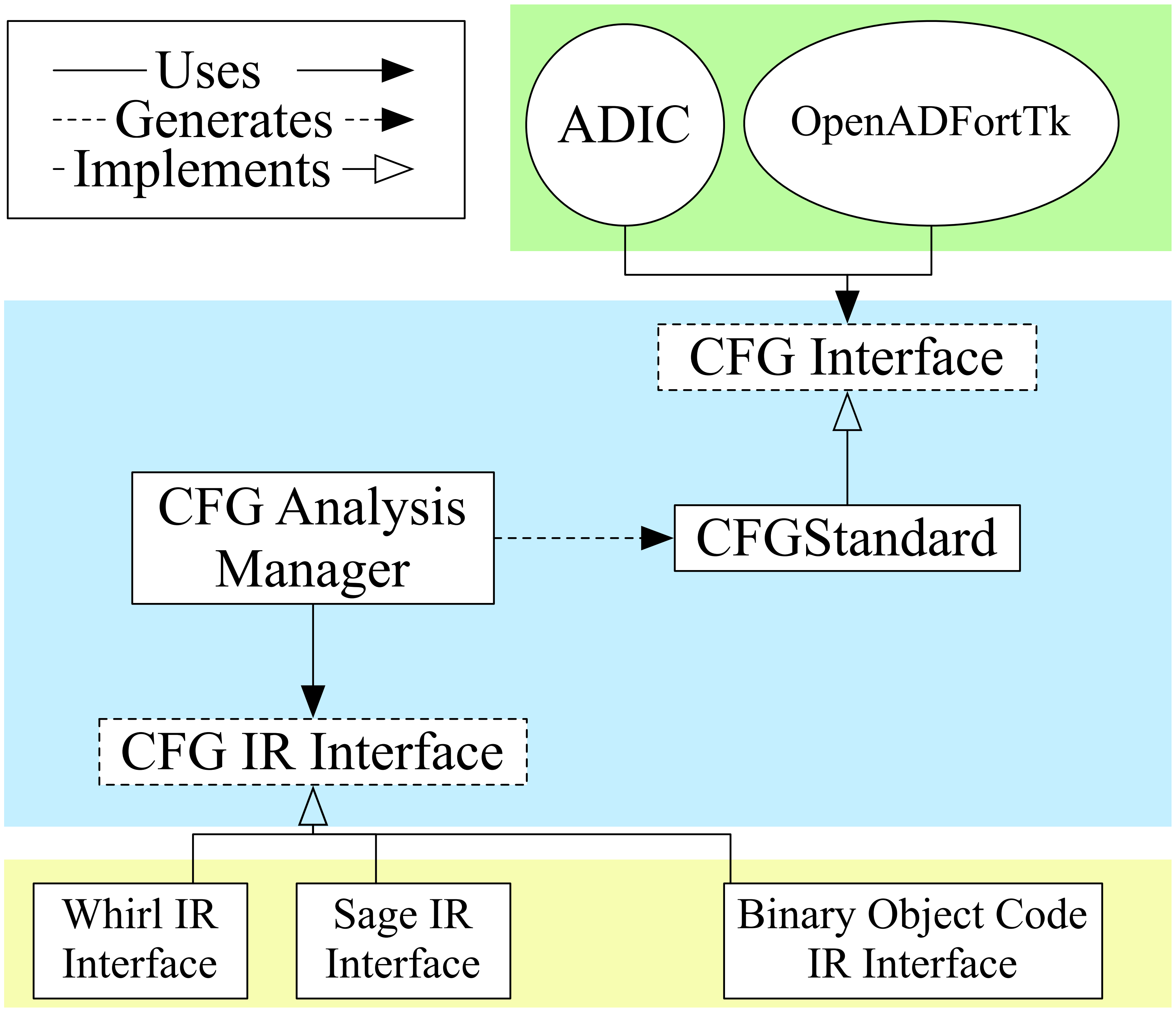
Michelle Strout

1/20/05

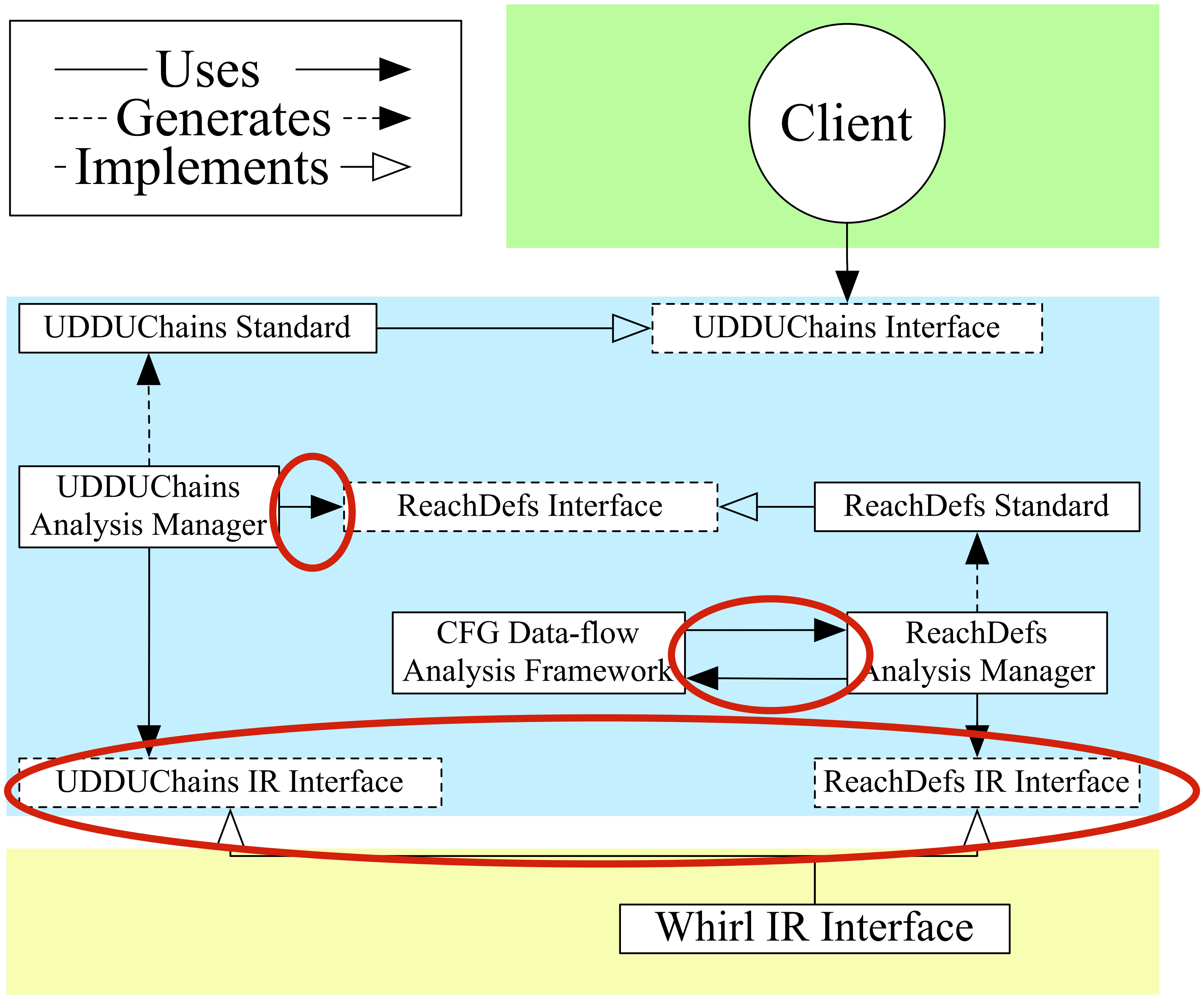
OpenAnalysis Overview



Control-flow Graph Example



Interacting Analyses and Analysis Frameworks



Opaque Handles to Source IR

- ProcHandle, StmtHandle, MemRefHandle, ExprHandle, etc.
- Provided by the source IR
- Analysis managers in OA provide analysis results associated with handles

General Approach for Developing an Analysis-Specific IR Interface

- Represent relevant program constructs with an opaque handle
- Make queries on handles for more information
- Example: Control-flow graph analysis

```
CFG::IRStmtType
  getCFGStmtType(StmtHandle)

SIMPLE, COMPOUND, LOOP,
STRUCT_TWOWAYCONDITIONAL, ...
```

Alias and Pointer Analysis

- Determines which memory references may or must reference the same program state (or memory locations)
- Important for many other analysis algorithms

Aliasing due to Arrays

```
REAL, dimension(10) :: A

A(0) = ...
do i = 1, 10
  A(i) = ...
end do
... = A(4)
```

- A(0) does not alias A(i) but won't detect until doing array section analysis
- A(i) does alias A(4)
- A(0) does not alias A(4)

Aliasing due to Reference Parameters

```
procedure foo(x,y)
integer,
intent(inout) :: x,y
...
end function


program bar
integer a

call foo(a,a)
```

- x and y are aliased in call to foo that happens in bar
- A reference parameter can also alias a global
- For now we plan to merge aliasing of all calls to same procedure

Aliasing due to Pointers

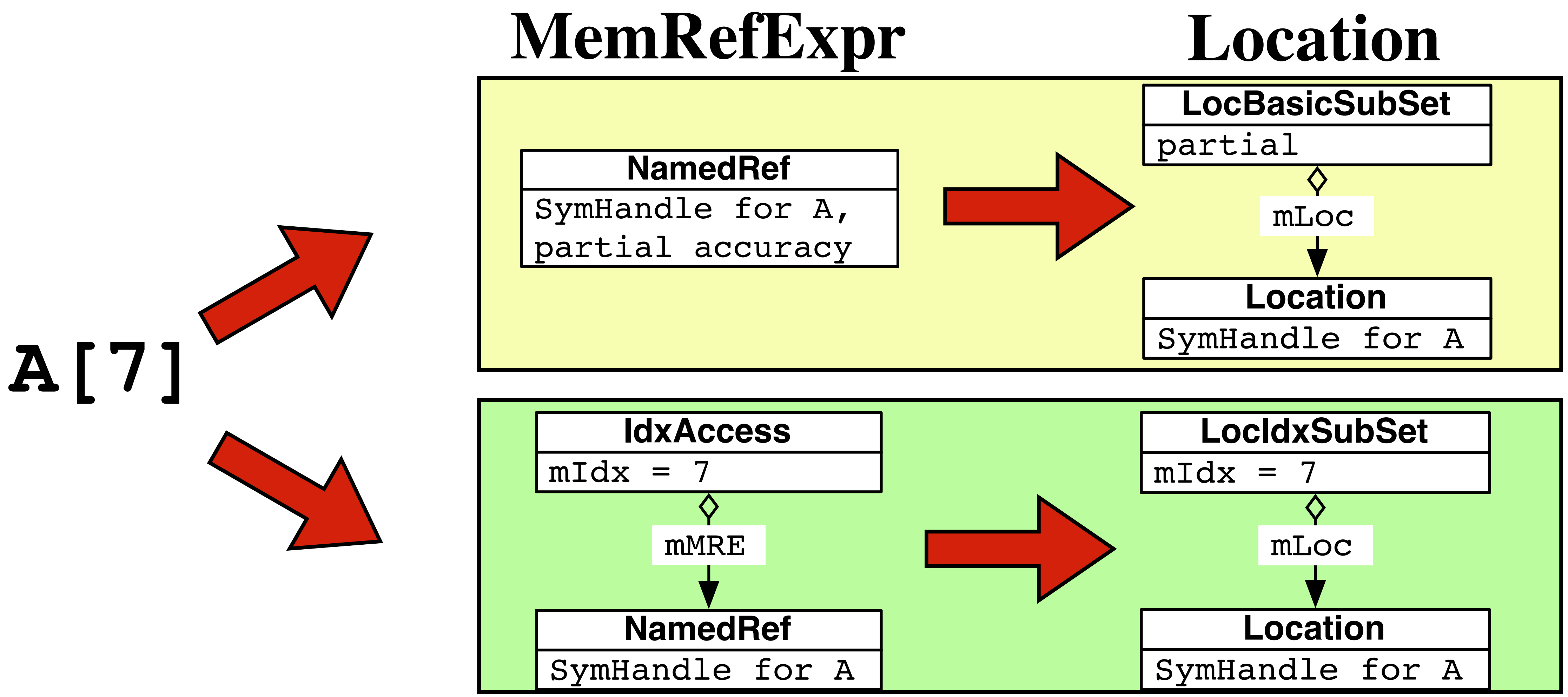
```
REAL, POINTER :: P
REAL, TARGET :: T1,T2
REAL :: G

if (flag)
  P => T1
else
  P => T2
end
G = P
```

- At G=P statement P may alias T1 or T2
- Current alias algorithm assumes pointers point to Unknown location
- Required interface to implement pointer analysis does exist

Approach to Determining Aliasing in OpenAnalysis

- query source IR for memory reference expressions that describe a memory reference
- map memory reference expressions to locations



Example Alias Analysis Results

- Analyze the set of may and must locations for each memory reference

```
REAL, POINTER :: P
REAL, TARGET :: T1, T2
if (flag)
   P => T1
else
   P => T2
end
G = P
```

Location
SymHandle for T1

or

Location
SymHandle for T2

Data-flow Analysis

- Operates on Locations
- Reaching Constants: if possible associates a constant value with locations
- Side-effect Analysis: keeps track of locations that may or must be modifiedor used

Status of IR Interfaces

- Whirl IR Interface (Open64IRInterface)
  - Have interfaces for CFG, CallGraph, Alias, Reaching Definitions, UDDUChains, Side-effect, and Activity
  - Still Needed
    - Persistent IR Handle values
    - Appropriate memory reference expressions for pointers
- Sage IR Interface (SageIRInterface)
  - Beata is working on generating memory reference expressions, which are a basic requirement for most analyses

Status of Analyses

- Analyses that have been converted to NewOA
  - CFG
  - CallGraph
- Analyses in implementation and testing stages
  - Alias analysis
  - UDDUChains
  - Interprocedural side-effect analysis
  - Activity analysis
  - Reaching constants
  - Activity analysis over MPI-CFG

Alias Analysis

- Aliasing due to aggregate types and arrays

```
A(i) = ...
x = ...
... = A(3)
```

Memory Reference	Alias Map Set	Locations
A(i)	1	{<0..1, partial>}
x	2	{<2..2, full>}
A(3)	3	{<1..1, full>}

- Status: implemented but seems to be broken for array references at the moment
- To handle constant array refs need more precise mem ref expressions and code to convert those to locations

Alias Analysis cont...

- Aliasing due to reference parameters
  - currently assume make optimistic assumption that aliases due to reference parameters don't happen
  - requires interprocedural alias analysis
  - working on the interprocedural piece right now
- Aliasing due to pointers
  - any dereference in a memory reference expression results in a mapping to the Unknown location
  - requires an alias algorithm that maintains a points-to datastructure

Use-def and Def-use Chains
UDDUChains aka. DU_UD

- For each use memory reference lists the statements that might define it. For each def memory reference lists the statements that might use it.
- Used to create computational graphs within basic blocks
- Jean, Nathan, and I have been actively debugging this for the past few weeks

Interprocedural Side-effect Analysis

- Determines the set of locations used (USE), definitely defined (DEF), possibly modified (MOD), and possibly referenced (REF) for the procedure
- Not affected by optimistic assumption about reference parameters not aliasing
- When translate to XAIF need to convert location abstraction to variable references, more work on this specification is need to enable implementation in whirl2xaif

Activity Analysis

- Interface for usage: provide an iterator for independent and dependent locations
- Analysis indicates active locations, active statements (those that may define an active location), and active memory references
- We need to discuss how independent and dependent locations will be specified in OpenADFortTk and how to represent results in XAIF
- Needs more testing

Better Testing Strategy

- IR Interface implementations
  - careful hand analysis to verify small examples of most cases and then regression testing
  - need persistent handles to compare results for regression testing
- Analyses
  - again careful hand analysis and then regression testing with examples from each source IR
  - have analysis-specific IR interface parsers for some but generating examples is time consuming